HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

**Project report:**

# HOUSE PRICE PREDICTION

**Guided by:**

Associate Professor Pham Van Hai

**Group members:**

Pham Quang Anh - 20220071

Vu Binh Minh - 20226058

Ngo Minh Trung - 20226004

Dang Trong Van - 20226072

Nguyen Long Vu - 20226006

# Contents

## Abstract

House prices are a crucial factor in the real estate market, impacting both buyers and sellers. Accurately predicting house prices can be a significant challenge due to the complex interplay of various factors. This paper explores the potential of artificial intelligent techniques for house price prediction. We present a review of existing research in this field, highlighting the commonly used machine learning algorithms such as linear regression, decision trees, random forest. We discuss the importance of data preprocessing, feature selection, and model evaluation in achieving reliable predictions.

## 1 Introduction

### 1.1 Context

House price prediction plays a significant role in making informed decisions for various stakeholders within the real estate market. It helps navigate the complexities of the market and fosters a more stable and efficient environment.

### 1.2 Motivation

House price prediction offers significant value to various stakeholders in the real estate market. By enabling informed decisions, improved strategies, and increased market transparency, house price prediction plays a crucial role in a healthy and efficient housing market.

## 2 Methodology

### 2.1 ElasticNet Regression

ElasticNet is a regularization technique that combines both L1 and L2 regularizations to handle multicollinearity and select important features. While L2 uses the square value, L1 uses the absolute value. L1 has an interesting property that it can shrink the weights to 0, making useless parameters not included in the model.

ElasticNet is a combination of both L1 and L2. It is particularly useful when the parameters are associated with the correlated variables, the model can shrink these parameters or remove them all at once.

$$\mathcal{L}(\mathbf{w}) = \underbrace{\boxed{\frac{1}{N}\|\bar{\mathbf{X}}\mathbf{w}-\mathbf{y}\|_2^2}}_{\text{sum of squared residuals}} + \underbrace{\boxed{\alpha\,[\,\lambda\|\mathbf{w}\|_1}}_{\text{L1 regularization (Lasso)}} + \underbrace{\boxed{\frac{(1-\lambda)}{2}\|\mathbf{w}\|_2^2\,]}}_{\text{L2 regularization (Ridge)}}$$

## 2.2 Decision Tree

Decision Tree is a tree-like structure where each node represents a feature, and the branches represent the decision rules. Consider a regression problem where we want to predict the price of a car based on its mileage and age.

We build a tree by selecting a feature, trying different threshold values and compare them by a metric (usually MSE).

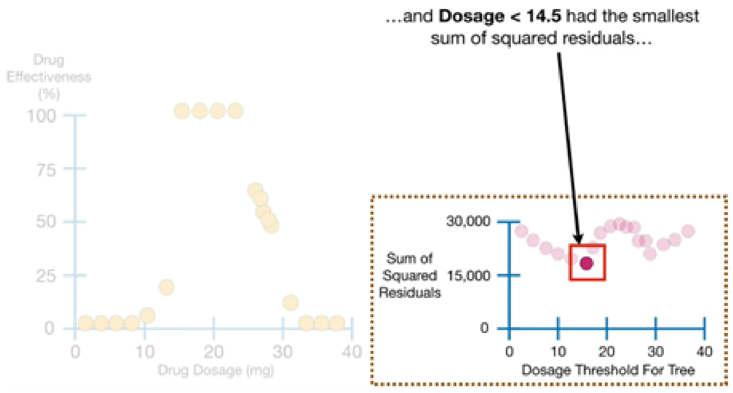
For example, for the CART (Classification And Regression Tree), the Variance Reduction metric is used, defined as the total reduction of the variance of the target variable $Y$ due to the split at this node:

$$I_V(N) = \frac{1}{|S|^2}\sum_{i\in S}\sum_{j\in S}\frac{1}{2}(y_i-y_j)^2 - \left(\frac{|S_t|^2}{|S|^2}\frac{1}{|S_t|^2}\sum_{i\in S_t}\sum_{j\in S_t}\frac{1}{2}(y_i-y_j)^2 + \frac{|S_f|^2}{|S|^2}\frac{1}{|S_f|^2}\sum_{i\in S_f}\sum_{j\in S_f}\frac{1}{2}(y_i-y_j)^2\right) \tag{1}$$

where $S, S_t$, and $S_f$ are the set of pre-split sample indices, set of sample indices for which the split test is true, and set of sample indices for which the split test is false, respectively. The expression inside the bracket is the weighted-sum of variance after spliting, so the formula can be interpreted as

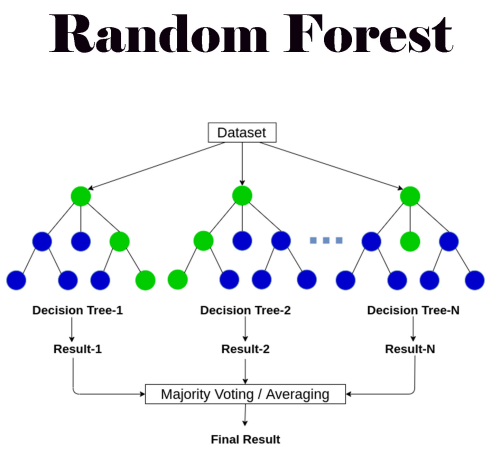
$$I_V(N) = \text{Variance before split} - \text{Weighted variance after split}$$

If we have multiple features, we compare the best threshold values for each of them. After that, select the best threshold to build a node and continue until a leaf node meets our preset stopping condition.



## 2.3 Random Forest

The weakness of Decision Tree (DT) is that they tend to overfit the data and not perform well on the test set. Random Forest is an ensemble learning algorithm that combines the simplicity of DTs. It first creates a bootstrapped dataset (that is random sampling with replacement), then build a DT but using only a random subset of features at each level of the tree. Then this process repeats hundreds of times, creating a "forest" of DTs. Finally, the value is determined by taking the average of all results. The variety makes RF much more effective than individual trees.



## 2.4 Gradient Boosting

Gradient Boosting combines weak learners into a single strong learner iteratively. To explain the mechanism, it is good to start from a more simple problem like least-square regression. Its goal is to learn a predictive model **F(x)** by minimizing the Mean Square Error $\frac{1}{n}\sum_i(\hat{y}_i - y_i)^2$

Consider a gradient boosting algorithm with **M** stages, at step $m$ ($1 \leq m \leq M$), with the current model $F_m$. In order to improve $F_m$, our algorithm adds a new estimator $h_m(x)$ (usually this is a Decision tree).

$$F_{m+1}(x_i) = F_m(x_i) + h_m(x_i) = y_i \tag{2}$$

Or equivalently

$$h_m(x_i) = y_i - F_m(x_i) \tag{3}$$

From this equation, we see that the goal is to fit the estimator $h_m$ with the value $y_i - F_m(x_i)$. The loss function of this estimator is $L(y, F(x)) = \frac{1}{2}(y - F(x))^2$, we want to minimize $J = \sum_i L(y_i, F(x_i))$. Here, $F(x_i)$ are parameters, we calculate the derivatives:

$$\frac{\partial J}{\partial F(x_i)} = \frac{\partial \sum_i L(y_i, F(x_i))}{\partial F(x_i)} = \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} = F_m(x_i) - y_i \tag{4}$$

This result is similar to the gradient descent formula

$$F(x_i) := F(x_i) - \gamma\frac{\partial J}{\partial F(x_i)} \tag{5}$$

where $\gamma$ is the learning rate ($0 \leq \lambda \leq 1$)

Furthermore, we can optimize $\gamma$ by finding the $\gamma_m$ value for which the loss function has a minimum:

$$\gamma_m = \text{arg min} \sum_i L(y_i, F_m(x_i)) = \text{arg min} \sum_i L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i) \tag{6}$$

The new model formula:

$$F_{m+1}(x_i) = F_m(x_i) + \gamma_m h_m(x_i) \tag{7}$$

In general, more weak models $h_m$ are added as necessary to make the prediction more accurate over time.

## 2.5 XGBoost

XGBoost improves the "tree-building" part of Gradient Boosting, by several different methods:

- The tree is built based on the gain metric, which is also used to prune the tree (if the gain value does not sufficient, we can remove the node - "pruning")
- The loss function uses Newton-Raphson method instead of the regular Gradient.

  Recall the Gradient used in Gradient boosting:

$$\hat{g}_m(x_i) = \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \tag{8}$$

  Newton-Raphson method also requires the Hessian:

$$\hat{h}_m(x_i) = \frac{\partial^2 L(y_i, F(x_i))}{\partial F(x_i)^2} \tag{9}$$

  Similar to normal Gradient Boosting, the goal is to fit the estimator $\phi_m$ with the value $\frac{\hat{g}_m(x_i)}{\hat{h}_m(x_i)}$

  The new loss function to this problem is:

$$\hat{\phi}_m(x) = \frac{1}{2}\hat{h}_m(x_i)\left[\phi(x_i) - \frac{\hat{g}_m(x_i)}{\hat{h}_m(x_i)}\right]^2 \tag{10}$$

  Finally, update the model:

$$\hat{f}_m(x) = \hat{f}_{m-1}(x) + \alpha\hat{\phi}_m(x) \tag{11}$$

  This may sound complicated, but for MSE loss, this is reduced to something called the similarity score - the ratio of (sum of residuals) squared and number of residuals

- Regularization, by putting penalties to the Output Value

# 3 State of art models

Several studies ([1][2]) emphasize that the choice of model can influence data considerations. For instance, Random Forest generally handles various data types without extensive preprocessing, while models like Elastic Net (known for feature selection) might benefit from high-dimensional data containing many features.

Studies consistently report the strong performance of XGBoost in house price prediction tasks. This is attributed to its optimized algorithms and ability to handle complex feature interactions, as discussed in [2],[3]. The interpretability of models is a recurring theme in the literature. Linear Regression offers interpretability but may struggle with complex relationships ([2]). Random Forest and Gradient Boosting offer a balance, while XGBoost and can be challenging to interpret due to their ensemble nature or complex architectures ([1], [2], [3]).

Research suggests exploring methods to improve interpretability while maintaining prediction accuracy, particularly for models like Random Forest and XGBoost ([2, 3]). Further research on feature engineering techniques specifically tailored to house price prediction tasks can benefit various models ([3]). Developing more interpretable ANN architectures and improving training efficiency for house price prediction remain active areas of research (e.g., [2]).

# 4 Model

**Steps to create the model**

1. Import Libraries
2. Load Dataset
3. Data Cleaning
4. Exploratory Data Analysis
5. Feature Engineering
6. Dimensionality Reduction
7. Outlier Removal
8. Building a Model
9. Export model to pickle file
10. Deploy the User Interface

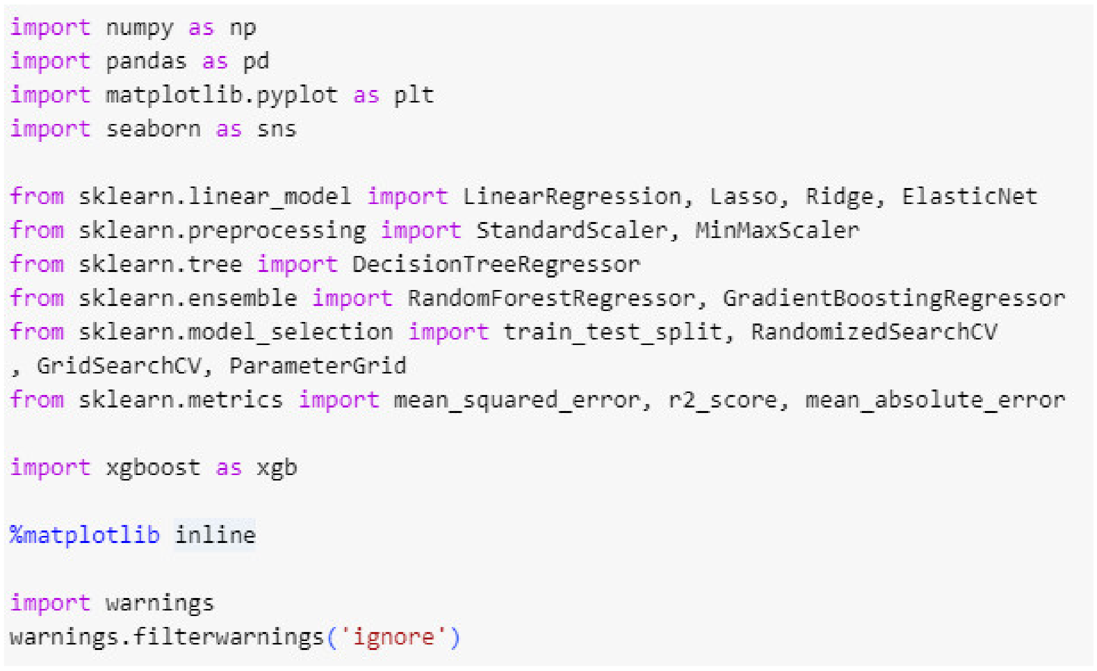
## 4.1 Import Libraries

Pandas - a data manipulation and analysis software package built-in Python. It specifically provides data structures and functions for manipulating numerical tables.

NumPy - NumPy is an abbreviation for Numerical Python, a package that is used to work with arrays. It also includes functions for working with linear algebra, Fourier transforms, and matrices.

Matplotlib and Seaborn - Matplotlib is a charting toolkit for the Python programming language and its numerical mathematics extension NumPy. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. Seaborn is used here to plot the correlation matrix.

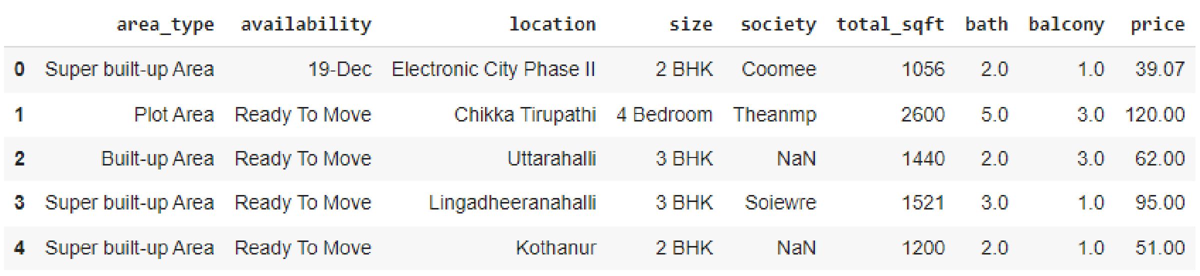
ScikitLearn and XGBoost - these libraries provide pre-build machine learning models, as well as also model-selection, allow us to choose the parameters set for optimal result.



```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.linear_model import LinearRegression, Lasso, Ridge, ElasticNet
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.model_selection import train_test_split, RandomizedSearchCV
, GridSearchCV, ParameterGrid
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

import xgboost as xgb

%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

## 4.2 Load Dataset

Dataset: `https://raw.githubusercontent.com/MSalah11GB/AIProject/main/Bengaluru_House_Data.csv`

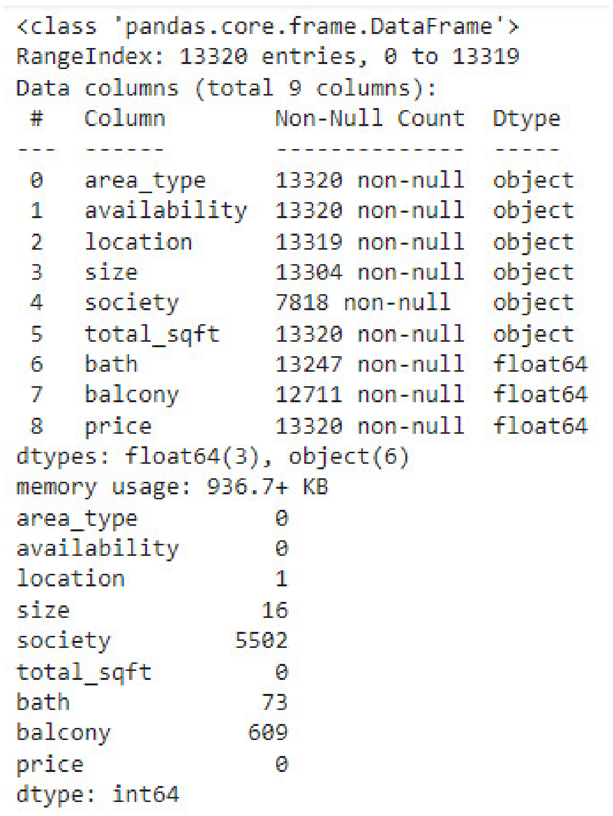


| | area_type | availability | location | size | society | total_sqft | bath | balcony | price |
|---|---|---|---|---|---|---|---|---|---|
| **0** | Super built-up Area | 19-Dec | Electronic City Phase II | 2 BHK | Coomee | 1056 | 2.0 | 1.0 | 39.07 |
| **1** | Plot Area | Ready To Move | Chikka Tirupathi | 4 Bedroom | Theanmp | 2600 | 5.0 | 3.0 | 120.00 |
| **2** | Built-up Area | Ready To Move | Uttarahalli | 3 BHK | NaN | 1440 | 2.0 | 3.0 | 62.00 |
| **3** | Super built-up Area | Ready To Move | Lingadheeranahalli | 3 BHK | Soiewre | 1521 | 3.0 | 1.0 | 95.00 |
| **4** | Super built-up Area | Ready To Move | Kothanur | 2 BHK | NaN | 1200 | 2.0 | 1.0 | 51.00 |

Here we use Pandas head() to show the first 5 datapoints. It is handy for rapidly determining whether your object contains the correct type of data.

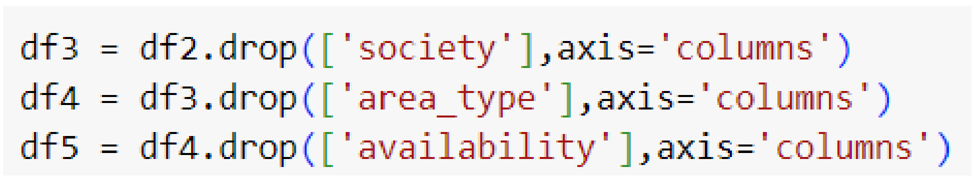
## 4.3 Data cleaning

.info() is used to collect meaningful insights about the data. .isnull().sum() finds the missing values



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13320 entries, 0 to 13319
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   area_type     13320 non-null  object 
 1   availability  13320 non-null  object 
 2   location      13319 non-null  object 
 3   size          13304 non-null  object 
 4   society       7818 non-null   object 
 5   total_sqft    13320 non-null  object 
 6   bath          13247 non-null  float64
 7   balcony       12711 non-null  float64
 8   price         13320 non-null  float64
dtypes: float64(3), object(6)
memory usage: 936.7+ KB
area_type          0
availability       0
location           1
size              16
society         5502
total_sqft         0
bath              73
balcony          609
price              0
dtype: int64
```

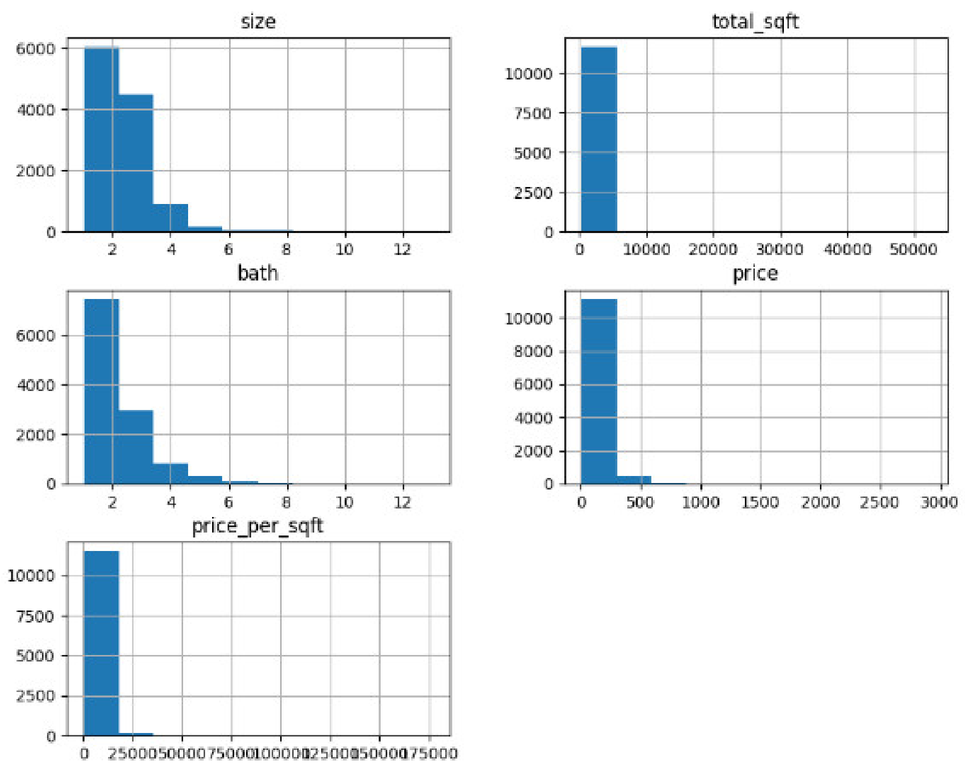
Drop the unnecessary features, or features that lack a large portion of key values.

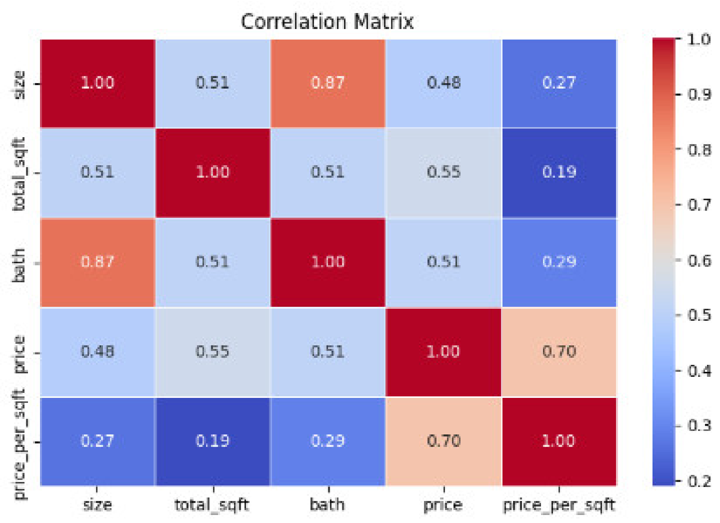


```
df3 = df2.drop(['society'],axis='columns')
df4 = df3.drop(['area_type'],axis='columns')
df5 = df4.drop(['availability'],axis='columns')
```

## 4.4 Exploratory data analysis

- **Skewness**: Almost all the distributions are heavily skewed to the right, indicating the presence of outliers or a non-normal distribution for these features.
- **Outliers**: There are significant outliers in the data, especially in the total_sqft, price, and price_per_sqft features. These outliers could be influential points and might need to be handled carefully during model training.
- **Data Concentration**: The majority of the data points are concentrated in lower ranges for all features, with only a small number extending to higher values. This might suggest that most of the properties in the dataset are relatively modest, with a few luxury properties.





|  | size | total_sqft | bath | price | price_per_sqft |
|---|---|---|---|---|---|
| size | 1.00 | 0.51 | 0.87 | 0.48 | 0.27 |
| total_sqft | 0.51 | 1.00 | 0.51 | 0.55 | 0.19 |
| bath | 0.87 | 0.51 | 1.00 | 0.51 | 0.29 |
| price | 0.48 | 0.55 | 0.51 | 1.00 | 0.70 |
| price_per_sqft | 0.27 | 0.19 | 0.29 | 0.70 | 1.00 |

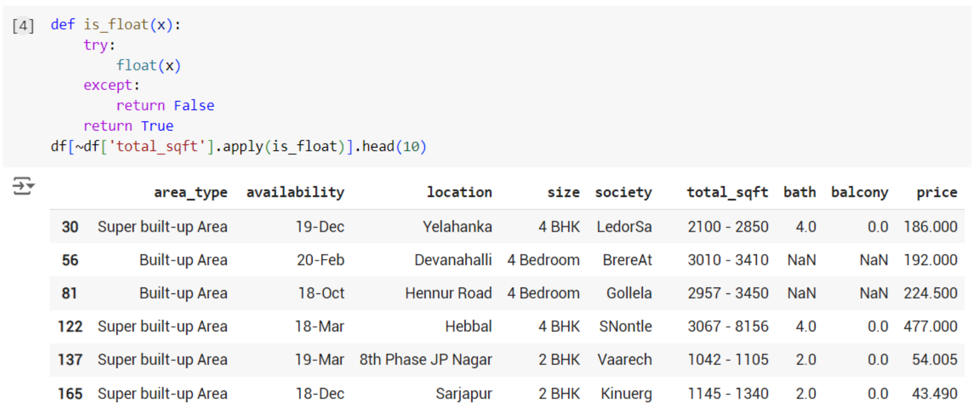
- **High Correlation Between Size, Total Square Footage, and Bathrooms**: The strong positive correlation between size, total_sqft, and bath suggests that larger houses tend to have more bathrooms and a larger total square footage.
- **Price Correlations:**
  - Price has moderate positive correlations with size, total_sqft, and bath, indicating that larger houses with more bathrooms and more total square footage tend to be more expensive.
  - The strong positive correlation between price and price_per_sqft suggests that higher-priced houses tend to have a higher price per square foot.

- **Price Per Square Foot**: The relatively lower correlations of price_per_sqft with size, total_sqft, and bath compared to their correlations with price suggest that while the price per square foot does increase with larger and more feature-rich houses, the relationship is not as strong as the absolute price.

## 4.5 Feature Engineering

Feature engineering is the process of creating new features from existing data, which is frequently dispersed over many linked tables. Feature engineering is collecting important information from data and consolidating it into a single table that can subsequently be utilized to train a machine learning model.

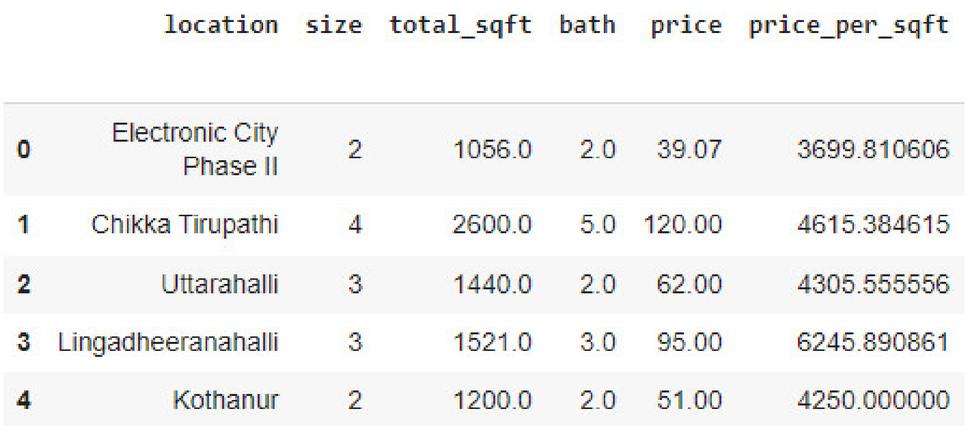
Explore the total_sqft feature:



```
[4] def is_float(x):
        try:
            float(x)
        except:
            return False
        return True
    df[~df['total_sqft'].apply(is_float)].head(10)
```

| | area_type | availability | location | size | society | total_sqft | bath | balcony | price |
|---|---|---|---|---|---|---|---|---|---|
| 30 | Super built-up Area | 19-Dec | Yelahanka | 4 BHK | LedorSa | 2100 - 2850 | 4.0 | 0.0 | 186.000 |
| 56 | Built-up Area | 20-Feb | Devanahalli | 4 Bedroom | BrereAt | 3010 - 3410 | NaN | NaN | 192.000 |
| 81 | Built-up Area | 18-Oct | Hennur Road | 4 Bedroom | Gollela | 2957 - 3450 | NaN | NaN | 224.500 |
| 122 | Super built-up Area | 18-Mar | Hebbal | 4 BHK | SNontle | 3067 - 8156 | 4.0 | 0.0 | 477.000 |
| 137 | Super built-up Area | 19-Mar | 8th Phase JP Nagar | 2 BHK | Vaarech | 1042 - 1105 | 2.0 | 0.0 | 54.005 |
| 165 | Super built-up Area | 18-Dec | Sarjapur | 2 BHK | Kinuerg | 1145 - 1340 | 2.0 | 0.0 | 43.490 |

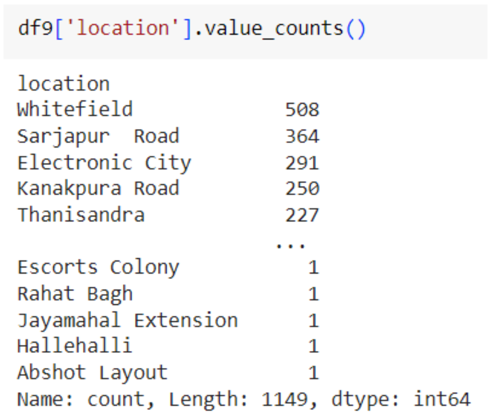
The above shows that total_sqft can be a range (e.g. 2100-2850). For such a case, we can just take an average of min and max value in the range.



| | location | size | total_sqft | bath | price | price_per_sqft |
|---|---|---|---|---|---|---|
| 0 | Electronic City Phase II | 2 | 1056.0 | 2.0 | 39.07 | 3699.810606 |
| 1 | Chikka Tirupathi | 4 | 2600.0 | 5.0 | 120.00 | 4615.384615 |
| 2 | Uttarahalli | 3 | 1440.0 | 2.0 | 62.00 | 4305.555556 |
| 3 | Lingadheeranahalli | 3 | 1521.0 | 3.0 | 95.00 | 6245.890861 |
| 4 | Kothanur | 2 | 1200.0 | 2.0 | 51.00 | 4250.000000 |

Add a new feature called price per square feet, it will be use later.

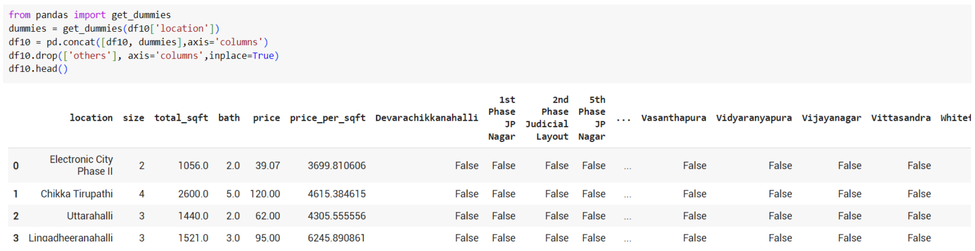
Examine the location feature, which are categorical labels. We need to apply the dimensionality reduction technique here to reduce the number of locations.



```
df9['location'].value_counts()
```

```
location
Whitefield             508
Sarjapur  Road         364
Electronic City        291
Kanakpura Road         250
Thanisandra            227
                      ...
Escorts Colony           1
Rahat Bagh               1
Jayamahal Extension      1
Hallehalli               1
Abshot Layout            1
Name: count, Length: 1149, dtype: int64
```

Finally, apply one-hot encoding to location feature.



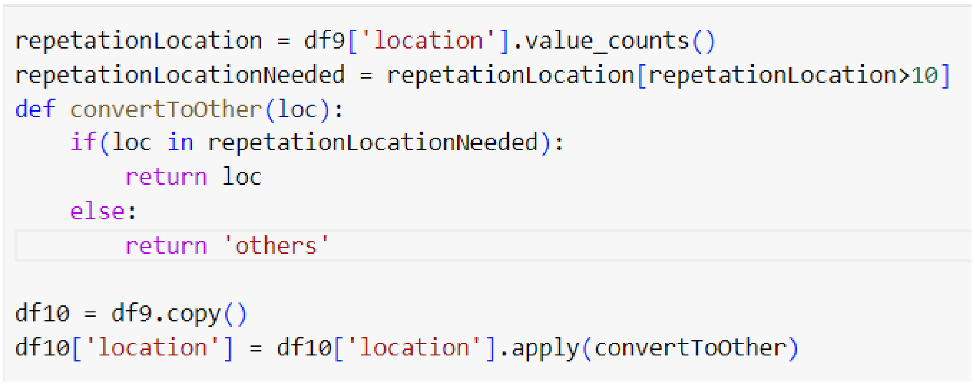
```
from pandas import get_dummies
dummies = get_dummies(df10['location'])
df10 = pd.concat([df10, dummies],axis='columns')
df10.drop(['others'], axis='columns',inplace=True)
df10.head()
```

| | location | size | total_sqft | bath | price | price_per_sqft | Devarachikkanahalli | 1st Phase JP Nagar | 2nd Phase Judicial Layout | 5th Phase JP Nagar | ... | Vasanthapura | Vidyaranyapura | Vijayanagar | Vittasandra | Whitef |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Electronic City Phase II | 2 | 1056.0 | 2.0 | 39.07 | 3699.810606 | False | False | False | False | ... | False | False | False | False | |
| 1 | Chikka Tirupathi | 4 | 2600.0 | 5.0 | 120.00 | 4615.384615 | False | False | False | False | ... | False | False | False | False | |
| 2 | Uttarahalli | 3 | 1440.0 | 2.0 | 62.00 | 4305.555556 | False | False | False | False | ... | False | False | False | False | |
| 3 | Lingadheeranahalli | 3 | 1521.0 | 3.0 | 95.00 | 6245.890861 | False | False | False | False | ... | False | False | False | False | |

## 4.6 Dimensionality Reduction

Dimensionality reduction methods are used to reduce the number of input variables in training data. When working with high-dimensional data, it is frequently beneficial to reduce the dimensionality by projecting the data to a lower-dimensional subspace that captures the data's "essence."

Any location that has less than 10 data points should be tagged as another location. This way the number of categories can be reduced by a huge amount. Later on, when we do one-hot encoding, it will help us with having fewer dummy columns.

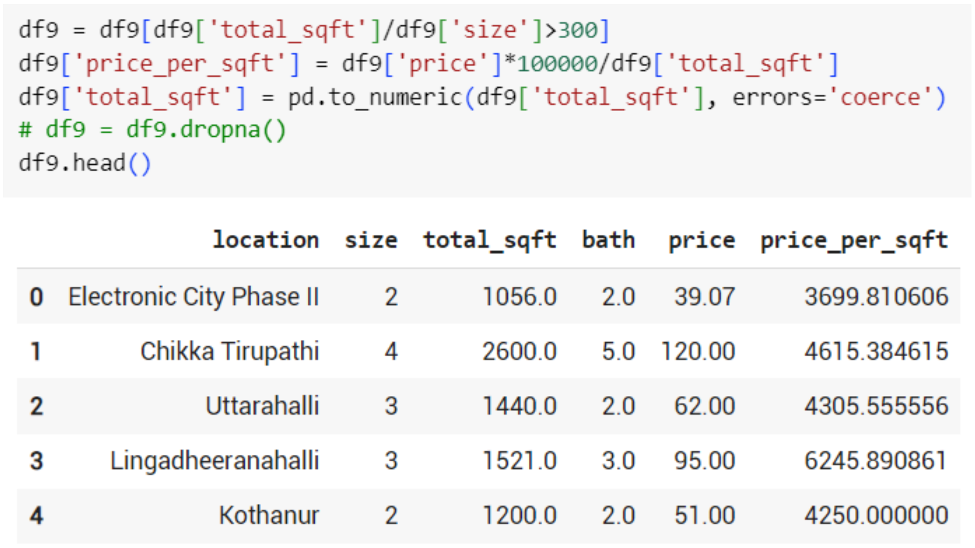


```
repetationLocation = df9['location'].value_counts()
repetationLocationNeeded = repetationLocation[repetationLocation>10]
def convertToOther(loc):
    if(loc in repetationLocationNeeded):
        return loc
    else:
        return 'others'

df10 = df9.copy()
df10['location'] = df10['location'].apply(convertToOther)
```

## 4.7 Outlier Removal

As a data scientist when you have a conversation with your business manager (who has expertise in real estate), he will tell you that normally square ft per bedroom is 300 (i.e. 2

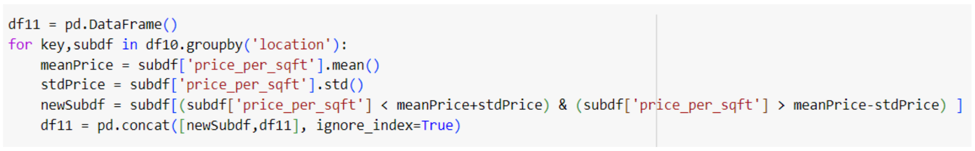
BHK apartment is minimum 600 sqft. If you have for example 400 sqft apartment with 2 BHK then that seems suspicious and can be removed as an outlier. We will remove such outliers by keeping our minimum threshold per BHK to be 300 sqft.



```python
df9 = df9[df9['total_sqft']/df9['size']>300]
df9['price_per_sqft'] = df9['price']*100000/df9['total_sqft']
df9['total_sqft'] = pd.to_numeric(df9['total_sqft'], errors='coerce')
# df9 = df9.dropna()
df9.head()
```

|   | location | size | total_sqft | bath | price | price_per_sqft |
|---|---|---|---|---|---|---|
| 0 | Electronic City Phase II | 2 | 1056.0 | 2.0 | 39.07 | 3699.810606 |
| 1 | Chikka Tirupathi | 4 | 2600.0 | 5.0 | 120.00 | 4615.384615 |
| 2 | Uttarahalli | 3 | 1440.0 | 2.0 | 62.00 | 4305.555556 |
| 3 | Lingadheeranahalli | 3 | 1521.0 | 3.0 | 95.00 | 6245.890861 |
| 4 | Kothanur | 2 | 1200.0 | 2.0 | 51.00 | 4250.000000 |

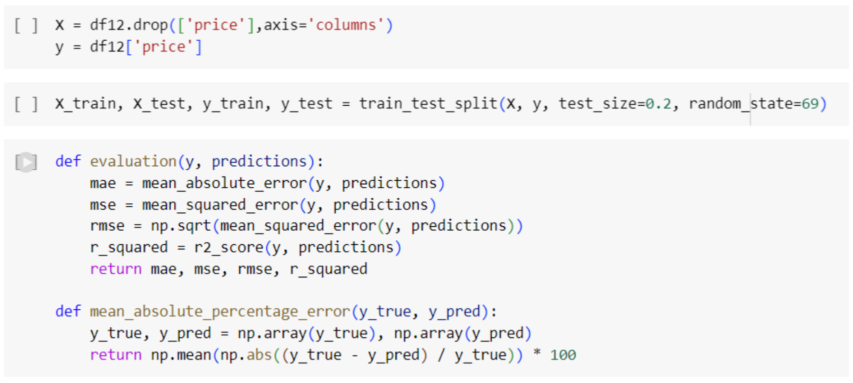
We also want to modify the price per sqft values of outliers by use the mean value of houses in the same location.



```python
df11 = pd.DataFrame()
for key,subdf in df10.groupby('location'):
    meanPrice = subdf['price_per_sqft'].mean()
    stdPrice = subdf['price_per_sqft'].std()
    newSubdf = subdf[(subdf['price_per_sqft'] < meanPrice+stdPrice) & (subdf['price_per_sqft'] > meanPrice-stdPrice) ]
    df11 = pd.concat([newSubdf,df11], ignore_index=True)
```

## 4.8 Building the Model

We split the dataset into train and test set and write function to print evaluation metrics.



```python
[ ] X = df12.drop(['price'],axis='columns')
    y = df12['price']
```

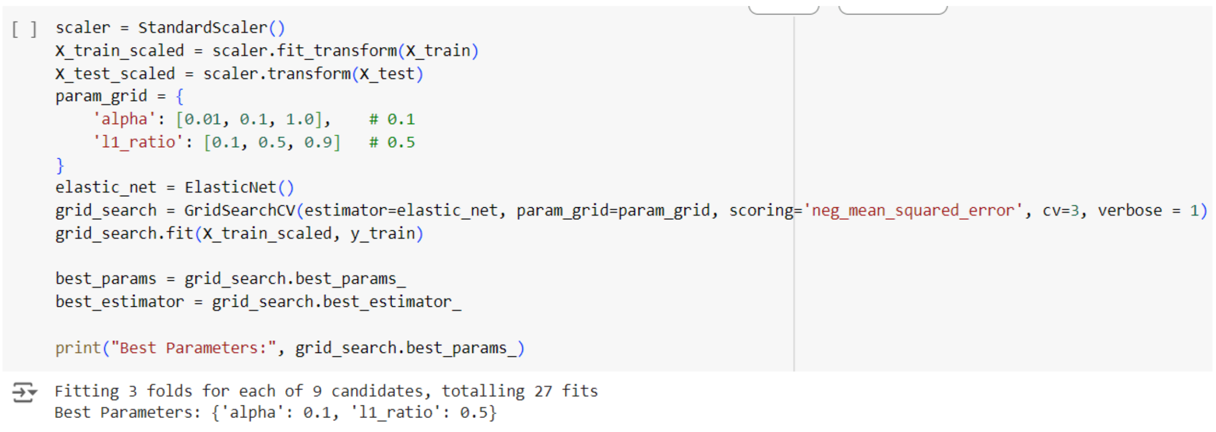
```python
[ ] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=69)
```

```python
def evaluation(y, predictions):
    mae = mean_absolute_error(y, predictions)
    mse = mean_squared_error(y, predictions)
    rmse = np.sqrt(mean_squared_error(y, predictions))
    r_squared = r2_score(y, predictions)
    return mae, mse, rmse, r_squared

def mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
```

For each model, we will use GridSearchCV to calibrate the parameters. To measure how good each model performs, we visualize the difference between the predicted value and test value, also known as 'residual'.

### 4.8.1 ElasticNet regression

Since models like Linear Regression are highly affected by the magnitude of inputs, we need to do an additional step of scaling the data.

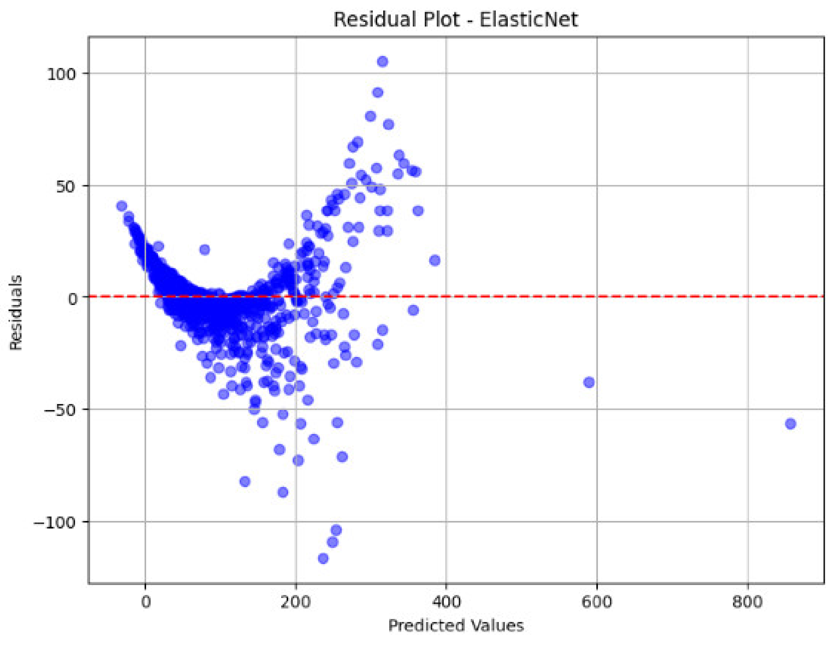


```
[ ] scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
    param_grid = {
        'alpha': [0.01, 0.1, 1.0],     # 0.1
        'l1_ratio': [0.1, 0.5, 0.9]    # 0.5
    }
    elastic_net = ElasticNet()
    grid_search = GridSearchCV(estimator=elastic_net, param_grid=param_grid, scoring='neg_mean_squared_error', cv=3, verbose = 1)
    grid_search.fit(X_train_scaled, y_train)

    best_params = grid_search.best_params_
    best_estimator = grid_search.best_estimator_

    print("Best Parameters:", grid_search.best_params_)
```

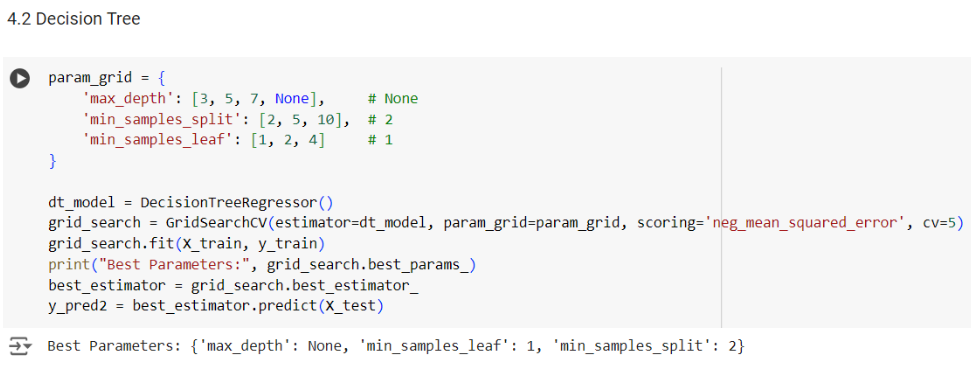
```
Fitting 3 folds for each of 9 candidates, totalling 27 fits
Best Parameters: {'alpha': 0.1, 'l1_ratio': 0.5}
```

Explanation of parameters:

- alpha: modifies the effect of regularization factors
- l1 ratio: modifies



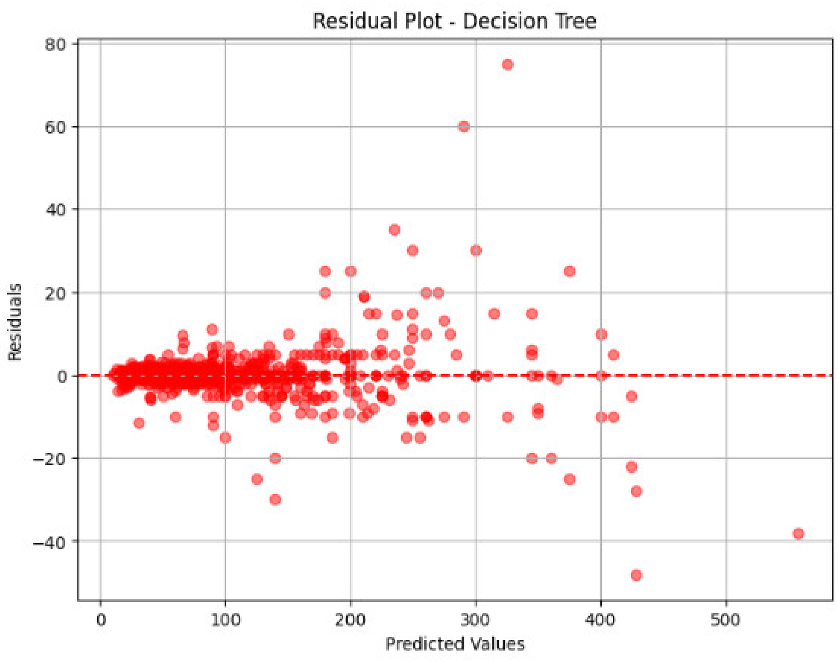
### 4.8.2 Decision Tree



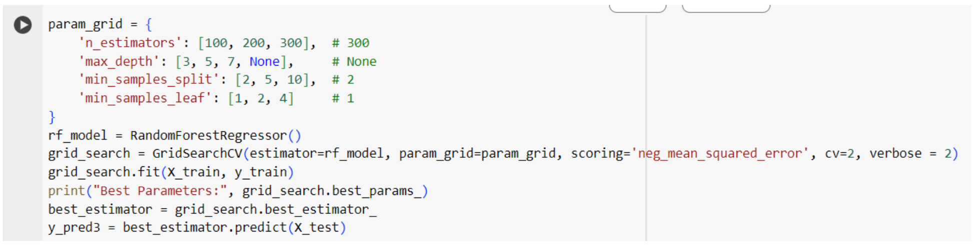
4.2 Decision Tree

```
param_grid = {
    'max_depth': [3, 5, 7, None],      # None
    'min_samples_split': [2, 5, 10],   # 2
    'min_samples_leaf': [1, 2, 4]      # 1
}

dt_model = DecisionTreeRegressor()
grid_search = GridSearchCV(estimator=dt_model, param_grid=param_grid, scoring='neg_mean_squared_error', cv=5)
grid_search.fit(X_train, y_train)
print("Best Parameters:", grid_search.best_params_)
best_estimator = grid_search.best_estimator_
y_pred2 = best_estimator.predict(X_test)
```

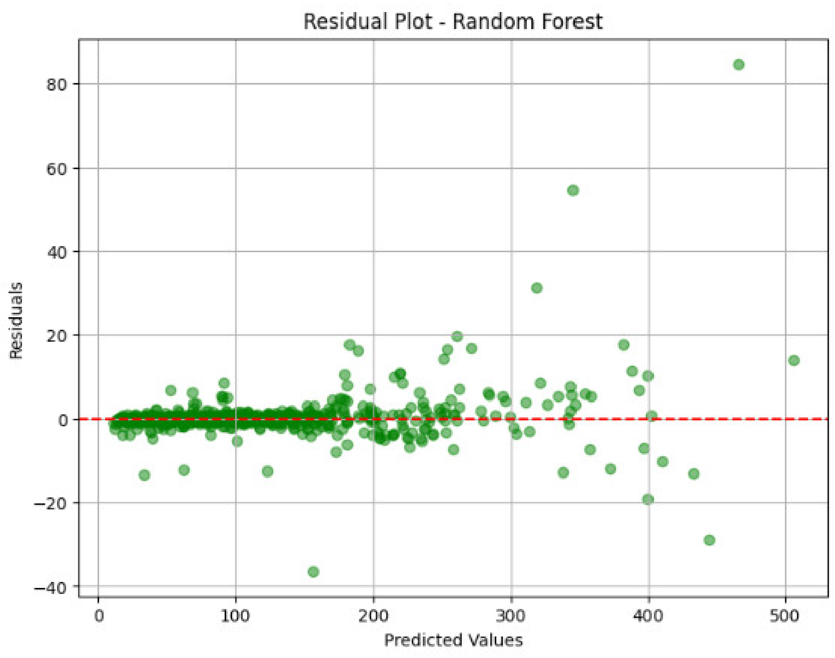
```
Best Parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2}
```



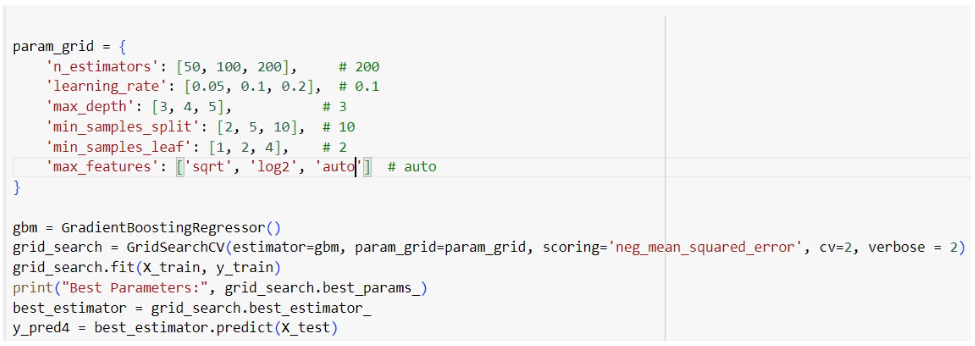
### 4.8.3 Random Forest



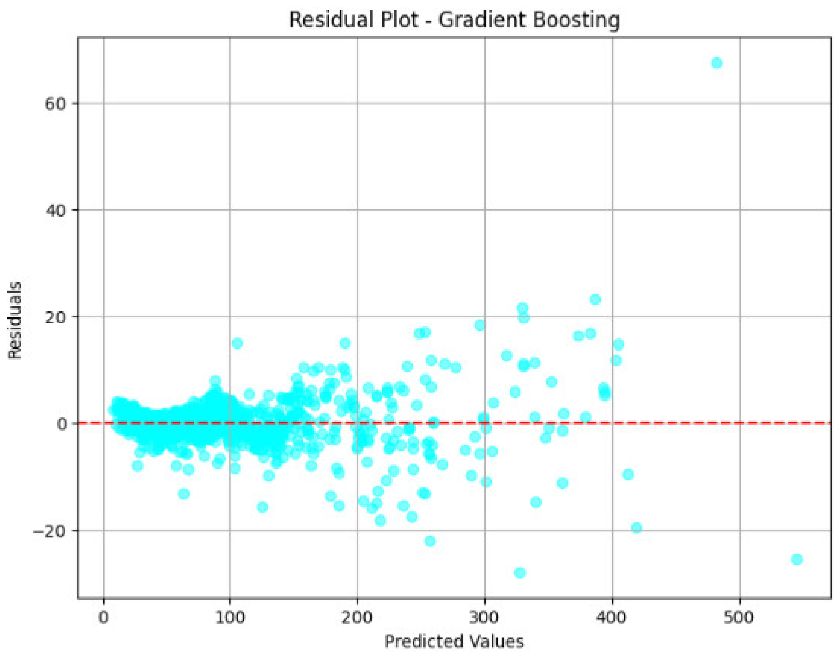
```
param_grid = {
    'n_estimators': [100, 200, 300],  # 300
    'max_depth': [3, 5, 7, None],      # None
    'min_samples_split': [2, 5, 10],  # 2
    'min_samples_leaf': [1, 2, 4]     # 1
}
rf_model = RandomForestRegressor()
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, scoring='neg_mean_squared_error', cv=2, verbose = 2)
grid_search.fit(X_train, y_train)
print("Best Parameters:", grid_search.best_params_)
best_estimator = grid_search.best_estimator_
y_pred3 = best_estimator.predict(X_test)
```



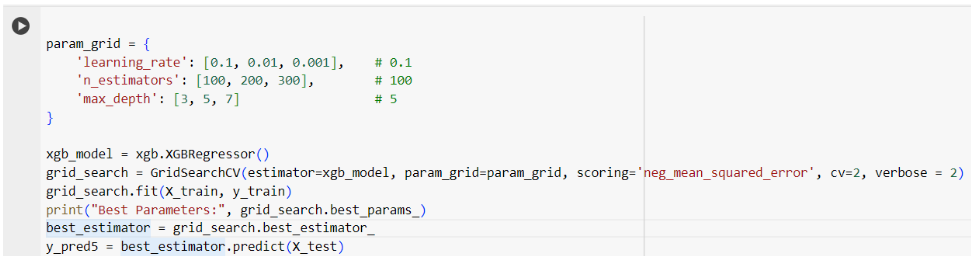
### 4.8.4 Gradient Boosting



```
param_grid = {
    'n_estimators': [50, 100, 200],     # 200
    'learning_rate': [0.05, 0.1, 0.2],  # 0.1
    'max_depth': [3, 4, 5],             # 3
    'min_samples_split': [2, 5, 10],    # 10
    'min_samples_leaf': [1, 2, 4],      # 2
    'max_features': ['sqrt', 'log2', 'auto']  # auto
}

gbm = GradientBoostingRegressor()
grid_search = GridSearchCV(estimator=gbm, param_grid=param_grid, scoring='neg_mean_squared_error', cv=2, verbose = 2)
grid_search.fit(X_train, y_train)
print("Best Parameters:", grid_search.best_params_)
best_estimator = grid_search.best_estimator_
y_pred4 = best_estimator.predict(X_test)
```



### 4.8.5 XGBoost



```
param_grid = {
    'learning_rate': [0.1, 0.01, 0.001],      # 0.1
    'n_estimators': [100, 200, 300],          # 100
    'max_depth': [3, 5, 7]                    # 5
}

xgb_model = xgb.XGBRegressor()
grid_search = GridSearchCV(estimator=xgb_model, param_grid=param_grid, scoring='neg_mean_squared_error', cv=2, verbose = 2)
grid_search.fit(X_train, y_train)
print("Best Parameters:", grid_search.best_params_)
best_estimator = grid_search.best_estimator_
y_pred5 = best_estimator.predict(X_test)
```

### 4.8.6 Evaluation for each model

Here are different metrics that we used in the evaluation process:

**Mean Absolute Error (MAE)** measures the average magnitude of the errors in a set of predictions, without considering their direction. It is calculated as the average of the absolute differences between predicted values and actual values:

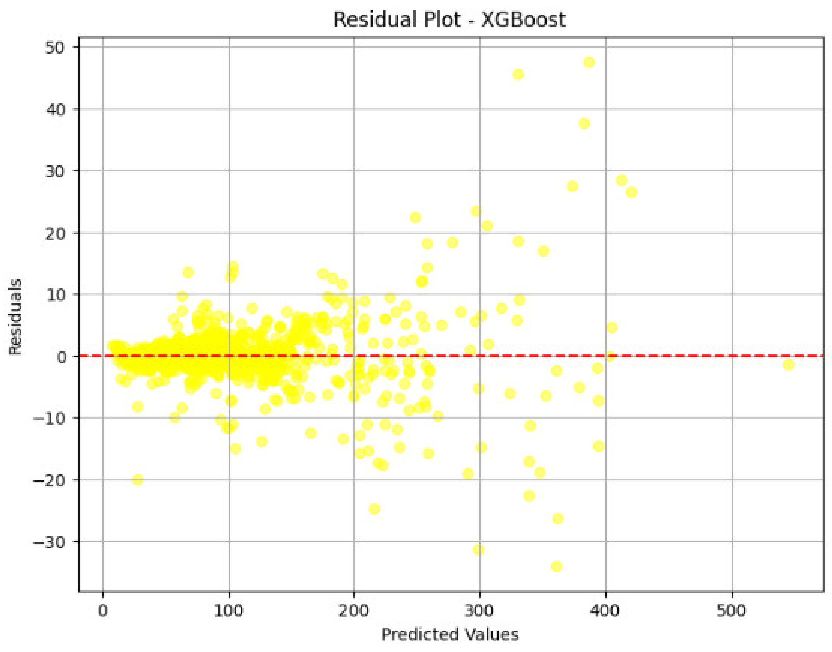
$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

where $y_i$ are the actual values and $\hat{y}_i$ are the predicted values.

**Mean Squared Error (MSE)** calculates the average squared difference between predicted and actual values. MSE gives more weight to large errors and is computed as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

**Root Mean Squared Error (RMSE)** is the square root of the MSE, providing a measure of the average magnitude of error in the same units as the target variable. It is



formulated as:

$$\text{RMSE} = \sqrt{\text{MSE}}$$

**R² score (R-squared)** represents the proportion of the variance in the dependent variable that is predictable from the independent variables. It is interpreted as the goodness of fit of the model and ranges from 0 to 1, with 1 indicating perfect predictions:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

where $\bar{y}$ is the mean of the observed data.

**Mean Absolute Percentage Error (MAPE)** measures the size of the error in percentage terms. It is particularly useful when the magnitude of errors is important and needs to be evaluated relative to the size of the actual values:

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \times 100$$

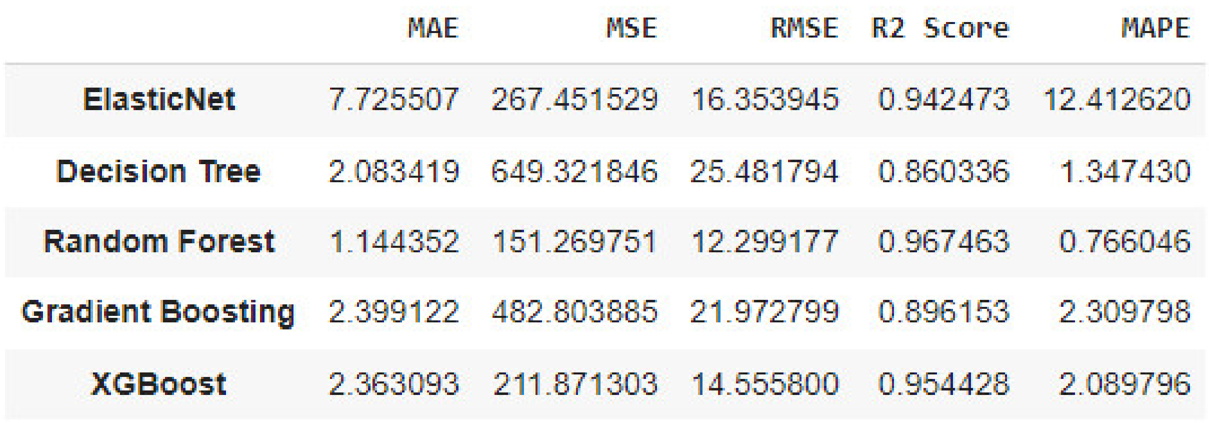
MAPE is expressed as a percentage, making it easier to interpret across different datasets.

These metrics play essential roles in evaluating predictive models, each offering unique insights into different aspects of model performance. The choice of metric depends on the specific goals of the analysis and the nature of the data being modeled.

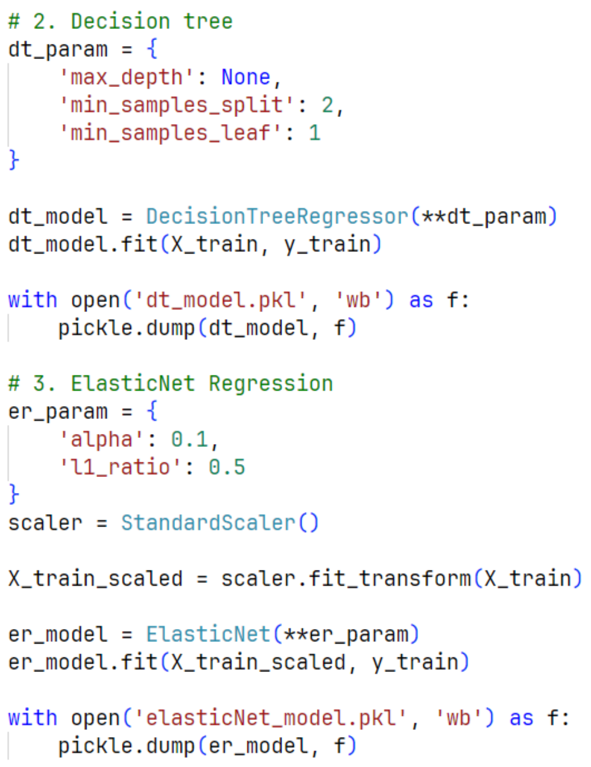
Overall, the Random Forest model performs the for all the evaluation metrics, while the ElasticNet performs the worst (possbibly due to the non-linearity of the data).

Interestingly, more complex models like Gradient Boosting and XGBoost did not perform better, even though it is one of the state-of-art algorithms. This may due to specific characteristics of the model, with little numerical features.

## 4.9 Export the model to pickle files



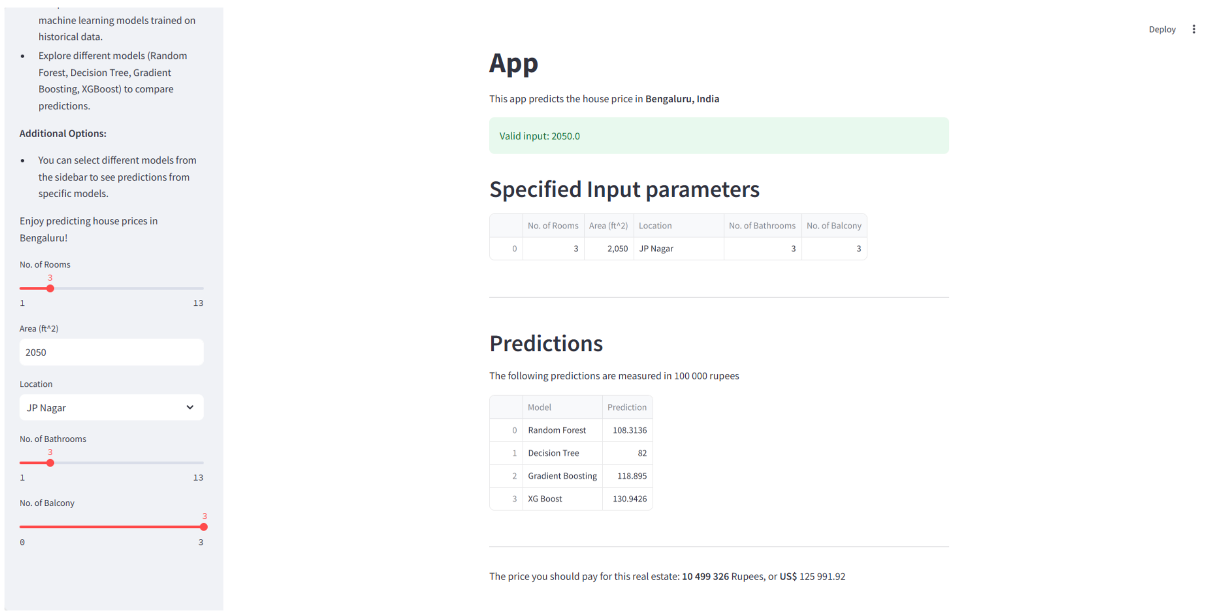
| | MAE | MSE | RMSE | R2 Score | MAPE |
|---|---|---|---|---|---|
| **ElasticNet** | 7.725507 | 267.451529 | 16.353945 | 0.942473 | 12.412620 |
| **Decision Tree** | 2.083419 | 649.321846 | 25.481794 | 0.860336 | 1.347430 |
| **Random Forest** | 1.144352 | 151.269751 | 12.299177 | 0.967463 | 0.766046 |
| **Gradient Boosting** | 2.399122 | 482.803885 | 21.972799 | 0.896153 | 2.309798 |
| **XGBoost** | 2.363093 | 211.871303 | 14.555800 | 0.954428 | 2.089796 |



```
# 2. Decision tree
dt_param = {
    'max_depth': None,
    'min_samples_split': 2,
    'min_samples_leaf': 1
}

dt_model = DecisionTreeRegressor(**dt_param)
dt_model.fit(X_train, y_train)

with open('dt_model.pkl', 'wb') as f:
    pickle.dump(dt_model, f)

# 3. ElasticNet Regression
er_param = {
    'alpha': 0.1,
    'l1_ratio': 0.5
}
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

er_model = ElasticNet(**er_param)
er_model.fit(X_train_scaled, y_train)

with open('elasticNet_model.pkl', 'wb') as f:
    pickle.dump(er_model, f)
```

## 4.10 Deploy the User Interface

For this project, we'll use StreamLit package from Python to build a simple User Interface. Details on how to deploy and use the model will be attached along with all the necessary files of the project.



machine learning models trained on historical data.

- Explore different models (Random Forest, Decision Tree, Gradient Boosting, XGBoost) to compare predictions.

**Additional Options:**

- You can select different models from the sidebar to see predictions from specific models.

Enjoy predicting house prices in Bengaluru!

No. of Rooms: 3 (1–13)

Area (ft^2): 2050

Location: JP Nagar

No. of Bathrooms: 3 (1–13)

No. of Balcony: 3 (0–3)

App

This app predicts the house price in **Bengaluru, India**

Valid input: 2050.0

Specified Input parameters

| | No. of Rooms | Area (ft^2) | Location | No. of Bathrooms | No. of Balcony |
|---|---|---|---|---|---|
| 0 | 3 | 2,050 | JP Nagar | 3 | 3 |

Predictions

The following predictions are measured in 100 000 rupees

| | Model | Prediction |
|---|---|---|
| 0 | Random Forest | 108.3136 |
| 1 | Decision Tree | 82 |
| 2 | Gradient Boosting | 118.895 |
| 3 | XG Boost | 130.9426 |

The price you should pay for this real estate: **10 499 326** Rupees, or **US$** 125 991.92

## 5 Conclusion

Predicting house prices with machine learning offers a powerful tool for navigating the complexities of the real estate market. While no model can guarantee perfect accuracy due to the ever-changing nature of the market and the influence of unforeseen factors, machine learning can significantly improve our ability to estimate value.

This exploration has highlighted the potential of various algorithms like linear regression, random forest, and support vector machines. By carefully selecting and evaluating these models on relevant datasets, we can identify the most effective approach for a specific market or property type.

## References

- Breiman, Leo; Friedman, J. H.; Olshen, R. A.; Stone, C. J. (1984). Classification and regression trees. Monterey, CA
- Exposition of gradient boosting by Cheng Li.
- Public dataset by Kaggle
- Literature Review on Real Estate Value Prediction Using Machine Learning Akshay Babu, Dr. Anjana S Chandran
- A Literature Review on Using Machine Learning Algorithm to Predict House Prices -Tanmoy Dhar; Manikandan P
- Prediction of House Price Using XGBoost Regression Algorithm - January 2021, Turkish Journal of Computer and Mathematics Education (TURCOMAT) 12(2):2151-2155